An Update on the RTI Design

Daniel J. Van Hook
MIT Lincoln Laboratory
244 Wood Street
Lexington MA 02173
(617)981-4153
dvanhook@ll.mit.edu

Stephen M. McGarry
MIT Lincoln Laboratory
244 Wood Street
Lexington MA 02173
(617)981-3776
smcgarry@ll.mit.edu

KEYWORDS

HLA, RTI, CORBA, Entity,Interoperability,Latency, Multicast,Network,Object Oriented, Scalability

ABSTRACT

The design and implementation of the RTI Prototype were described at the last Workshop. This paper will update Workshop participants on the anticipated evolution of the RTI design to support the different needs of HLA/RTI users.

INTRODUCTION

This document describes the functional design concept for the HLA RTI (High Level Architecture Run-Time Infrastructure) [1,2]. An overview of the prototype RTI design is given in reference [3]. The purpose of this document is to describe the functional design of the RTI post release 0.3. It is not a detailed software design per se, although it certainly takes software issues into account. The goal of this paper is to provide design disclosure for the library based RTI implementation.

FUNCTIONAL DESIGN

Figure 1 shows the functional design for the RTI. In figure 1, boxes indicate functional groupings. The partitioning shown in this figure does not necessarily map directly to software modules or to individual classes. Instead, the intent is to identify the functions that must be performed and their dependencies. Further, the interconnections shown in this figure do not necessarily map to function/procedure calls or to methods. Instead, lines indicate control and data flows that could be implemented via calls in either direction or access to common databases. The detailed software design of the RTI addresses these issues and is based on this functional design. The major features of Figure 1 are addressed in the following sections.

STREAM MANAGER

The stream manager (SM) provides transport services. Conceptually, it provides a virtual network layer that abstracts the underlying communications layer. The intent is that the SM will be sufficiently general so as to largely shield its clients from impacts related to changing communication services (e.g., direct use of ATM or of a shared memory interconnect). The initial assumption is that the communications layer will be IP.

The SM has no explicit notion of RTI data types such as objects and attributes. It treats all transported data as opaque byte arrays. This layering permits SM services to be used uniformly for object/attribute transport as well as other RTI communication. The SM does not and should not care what it is transporting.The following sections outline the primary abstraction supported by the SM – the stream – as well as the various SM transport services.

Stream Abstraction

The stream manager supports a communications abstraction called a *stream*. A stream has the following properties:

- *Multipoint to multipoint.* Each stream may have zero or more receivers and zero or more senders. Upper limits are imposed only by space, time, processing, and bandwidth limitations of the system.

- *Senders need not be receivers.* Senders to a particular stream are not required to also be receivers from that stream.
- *Connectivity only***.** A stream defines connectivity only. Other transport characteristics such as reliability are not defined by a stream. So, for example, different services providing different levels of reliable delivery may use the same connectivity defined by a single stream.

Stream Services

The stream manager supports two message delivery services: a reliable message delivery service and a best effort message delivery service.

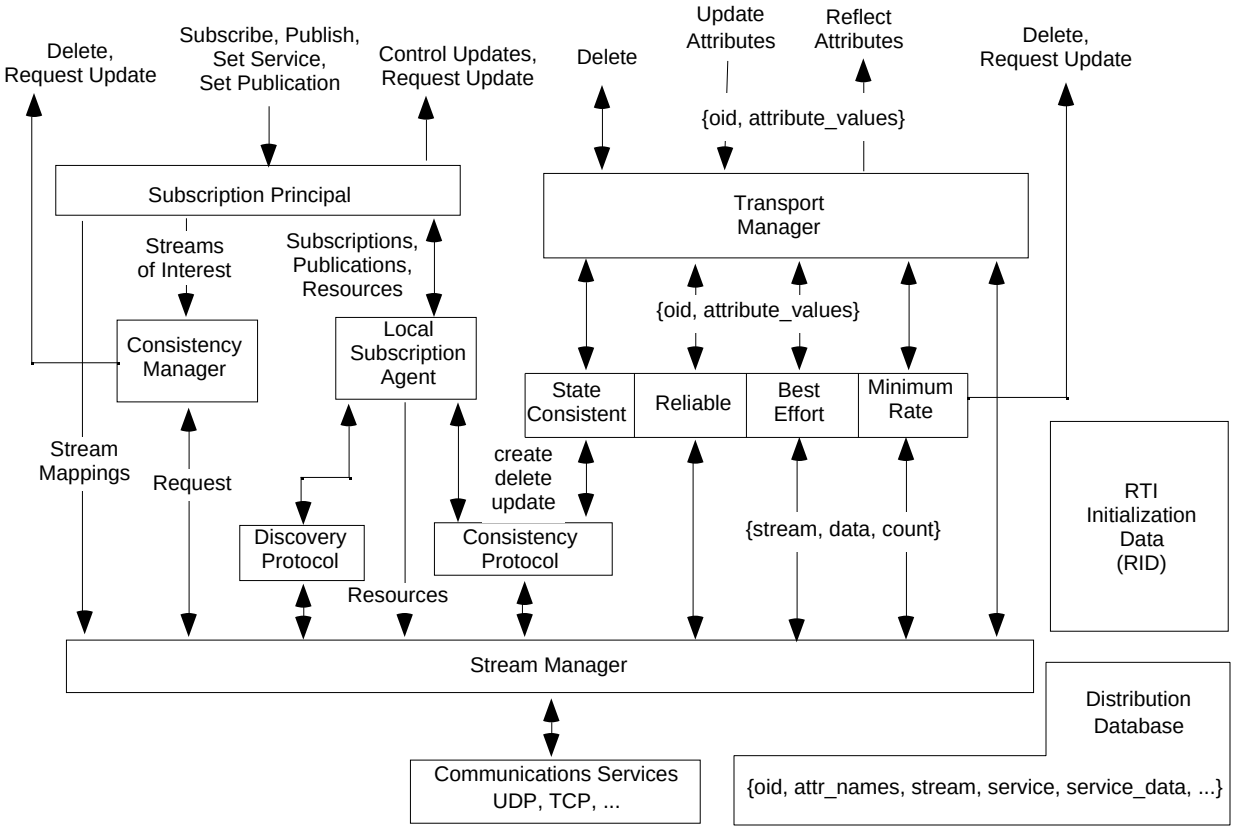

Figure 1 - RTI Functional Block Diagram

Reliable Message Service

The reliable message service provides reliable, ordered delivery of messages from a stream sender to all stream receivers. The data passed at the interface to this service consists of the following items:

- stream identifier
- byte array, i.e., a message
- count specifying the size of the byte array

The initial implementation of this service will be supported by TCP/IP connections. For this implementation, the SM will forward the data over each of the TCP connections in turn. Subsequent implementations may take advantage of reliable multicast protocols such as RAMP and RMP.

Best Effort Message Service

The best effort message service provides best effort message delivery from a stream sender to all stream receivers. No reliability mechanisms are employed, nor is any ordering guaranteed. The data passed at the interface to this service consists of the following items:

- stream identifier
- byte array, i.e., a message
- count specifying the size of the byte array

The initial implementation of this service will be supported by UDP/IP multicast and unicast transmission.

Stream Manager Processing

The SM performs the following processing.

Connectivity establishment. The SM receives stream mappings from the subscription principal. Stream mappings consist of:

- *Stream to multicast group associations.* Multicast groups are represented in an abstract fashion at the SM interface, i.e., they are not necessarily actual network addresses.
- *Stream to communication endpoint list associations.* Communication endpoints contain sufficient information to define a communicating entity, e.g., a host name and port number pair in the case of IP. Communication endpoints are represented as an opaque data type at the SM interface.
- *Stream direction.* Indication of whether the stream is for sending, receiving or both.
- *Message service type.* Whether the stream is used for best effort delivery, reliable delivery, or both.

The SM uses stream mappings to establish the desired connectivity. For example, the SM may join one or more multicast groups or it may open one or more TCP/IP connections, depending on the stream mappings it receives. In addition, the SM also maintains a forwarding table of stream to group or stream to connection mappings used when forwarding data.

Message forwarding. The SM forwards each message using the connectivity that has been established via the stream mappings. For each message it is called upon to transmit, the SM performs a forwarding table look up based on destination stream. The look up yields the details of what group to send to (in the case of the best effort service) or which set of connections to send to (in the case of the reliable service). Messages are encapsulated in an SM header.

Message receiving and filtering. The SM receives and filters messages. Filtering is based on stream identifier and the federation execution identifier. Received packets that are destined for streams for which the SM does not have a stream mapping or that are not from the federation that is joined are discarded. This filtering capability permits multiple streams to be multiplexed over a smaller number of multicast groups or TCP/IP connections.

Network byte ordering. The SM performs conversions as needed to access portions the headers with which it encapsulates client messages. The SM does not apply any format conversions to the messages themselves.

Resource reservation. The SM accepts resource reservations in terms of streams and reserves resources using the RSVP API supported by the communications layer.

TRANSPORT MANAGER

The transport manager (TM) manages the RTI's transport services. The current RTI transport services are:

- Best effort.
- Minimum rate.
- State consistent.
- Reliable.

The transport manager performs the following processing:

Attribute update. The TM accepts a list of attribute values to be transported. All attributes must be associated with the same object. The object is identified by an object identifier. The TM ascertains the transport service for each attribute value from the distribution database and conveys each attribute value to the specified service. In essence, the TM demultiplexes a list of attribute values into sublists destined for each transport service.

Attribute reflection. The TM accepts a list of attribute values received by a transport service. The TM conveys the attribute values to its client. In essence, the TM multiplexes attribute values received from the various transport services into a single path. The client federate does not know which transport service was employed to actually transmit the attributes. All attribute values received from a particular service must be associated with a single object. Each object is identified by an object identifier.

Fragmentation/reassembly. The TM fragments large messages before transmission for the best effort service. The fragment size is based on the MTU (maximum transmission unit) obtained from the underlying communications layer. The SM reassembles received messages before passing them to its client. Clients of TM may choose to do their own fragmentation/reassembly in order to optimize retransmission strategies. This may make sense in some cases (e.g., the consistency protocol) since TM does not by itself support an retransmission. It just makes use of reliable services of the underlying communications layer. To support client fragmentation/reassembly, TM makes the MTU available to its clients. Client fragmentation/reassembly is transparent to TM.

Transport Services

The transport services provide different types of reliable message delivery. All transport services share a common interface that shields the transport manager or the stream manager (i.e., the clients) from knowledge of internal protocol processing. The transport manager views the transport services as sources and sinks for lists of attribute values for particular objects. The stream manager views the transport services as sources and sinks for opaque byte arrays.

Processing common to all transport services consists of the following:

Message formatting. Formatting of the internal attribute representation into the externally visible message format.

Data format conversion. Conversion to and from machine and compiler specific data formats. This processing is done through XDR.

A particular attribute value may be delivered multiple times by a transport service under some conditions. In other words, no guarantee is made that an attribute value will be delivered only one time, even though the federate may have updated that attribute only once. Reliability mechanisms employed by one or more of the transport services may result in redundant reception and processing of the same attribute value. This design choice has been made to avoid the burden of tracking the version of every attribute that has been passed to the federate.

Specific characteristics of each transport service are described in the following sections.

Best Effort Service

The best effort (BE) transport service transmits attribute values with no reliability mechanism. In addition, messages may be received out of order. The best effort stream manager service is used to support BE. This service is available to support construction of application protocols or transport over lossless media such as shared memory.

Minimum Rate Service

The minimum rate (MR) transport service most closely emulates DIS transport mechanisms. Attributes are sent at a minimum rate even if no attribute values have changed. The stream manager's best effort stream service is used to support MR. This approach is a relatively crude consistency mechanism that compensates for late joiners and lost packets. It is an efficient and appropriate service for attributes that change value fairly frequently.

Processing performed by MR includes the following:

Minimum rate transmission. MR ensures that attribute values are sent at a minimum rate. If the federate has not updated all attributes for a given object published to a particular stream within a RID-determined interval, MR invokes the federate's request service. MR does not store previously sent attribute values. Instead it relies on the federate to update the attribute values as needed. MR maintains timers and data structures in the distribution database to keep track of the last transmission time for each set of attributes for each object publishing to a given stream and using the MR service.

Load leveling. Requests to the federate for attribute value updates are evenly distributed across the timeout interval using a round-robin strategy. This reduces the peak packet rate from federates immediately answering these requests.

Object deletion. MR will time out and delete objects for which no attributes have been received after a period of time. MR will invoke the federate's delete service via the object manager (discussed later) to notify the federate. The object manager in turn will remove all references to the deleted object from the distribution database. This capability can be enabled/disabled via the RID.

State Consistent Service

The state consistent (SC) transport service ensures delivery of the latest attribute values. It does not guarantee delivery of any intermediate attribute updates, however. The SC service is appropriate for attribute values that change infrequently.

The core of the SC service is implemented by the consistency protocol (CP). SC essentially forms a layer on top CP. The purpose of this layering is to maintain CP as a generic service available for other transport needs besides attribute updates. SC processing includes the following:

Data item management. SC maintains the set of attributes for each object publishing to a stream as a CP data item. Data items are created and deleted and their values are kept consistent across all relevant receivers using CP capabilities. SC can invoke the federate's request service to request an update for specified attributes.

Object deletion. A client's delete service is invoked via the object manager to delete an object if a data item maintained by CP times out. This capability can be enabled/disabled via the RID.

Final updates. Attribute value updates may be requested from the client multiple times by SC to boost reliability.

Reliable Service

The reliable (RE) transport service supports delivery of all attribute values to all receivers in the order sent. RE uses the reliable message service supported by the stream manager.

The early implementation of the reliable service will be a hierarchical network of server processes that maintain connections to RTI clients. The server accepts messages from a federate stream manager and delivers it reliably to each of the other participating federates. The federates discover the servers in the federation using the discovery protocol discussed later in this paper.

OBJECT MANAGER

The Object Manager helps to maintain the internal consistency of the RTI for special cases of client object instantiation and deletion. The Object Manager (OM) interfaces with the distribution database, subscription princpal, and transport manager.

Object instancing. Entries in the distribution database are created due to initial attribute updates for an object.

Object deletion. If the client federate indicates that a local object should be deleted, the OM transmits a delete message to the stream(s) to which the object is being published. The delete is transmitted using either the stream manager best effort or reliable service. If the OM receives a delete message from the stream manager, the OM informs the client that the object should be deleted. In addition, the OM also removes all entries associated with the object from the distribution database.

The subscription principal triggers distribution database object instantiation and deletion as groups are joined or left. The transport manager triggers distribution database object instantiation when attribute

updates for previously unlisted objects are either handed down from the local federate or received across the RTI from a remote federate. The minimum rate transport service also triggers object deletion when the minimum rate receive timeout expires.

SUBSCRIPTION PRINCIPAL AND AGENT

The subscription principal (SP) and subscription agent (SA) manage a federate's subscriptions and publications. The agent concept has been previously described in reference [4] and the subscription concept in reference [5].

Subscription Principal

The SP coordinates setting up the connectivity needed to transfer all the relevant and as little irrelevant simulation data as possible. The SP does not transfer attribute values itself. Its role is limited to arranging for connectivity. Attribute value transfer is the responsibility of the transport manager.

Processing performed by the SP includes:

Subscription and publication. Subscriptions and publications are expressed in terms of filter spaces [6]. Subscriptions and publications are referenced by handles.

Set service. The federate can set each attribute's transport service type on a per object basis. Default transport service types for each attribute are specified in the RID.

Set publication. A publication is associated with each attribute on a per object basis. The SP uses an attribute's publication to select a stream for the attribute. This data is maintained in the distribution database.

Subscription combination. Subscriptions may be combined by the SP as an optimization.

Stream calculation. The SP uses its local SA to calculate the streams that data must be received from and sent to.

Stream mapping. The SP conveys stream mappings to the stream manager. The stream manager uses stream mappings to set up the necessary connectivity and type of service. Stream mappings associate streams with multicast groups, lists of communication endpoints, and type of service (reliable or best effort).

Stream database. The SP maintains a database of streams of interest. Streams of interest are those streams from which the federate needs to receive data. This database is updated as the streams of interest change. This database is available for use by the consistency manager.

Control updates. The SP notifies the federate about what attributes are not of interest to any other federate. The federate need not update these attributes. Should any of these attributes later become interesting, the federate will be notified and should begin updating.

Request on stream change. The SP requests that the federate update attributes whose publication stream has changed.

Resource reservation. The SP provides communication resource requirements and limitations for each publication/subscription (e.g., bit and packet rates) to the local SA. The RID contains estimates of these parameters for each attribute. Future implementations may support dynamic run-time specification of resource requirements as well as dynamic estimation.

Subscription Agent

The SA calculates the streams that provide the connectivity to convey data from publishers to subscribers. Subscription agents may be local or remote. All distribution managers have a local SA and may utilize one or more remote SAs. The local SA is always available and provides stream calculation services for time critical data or in the absence of remote SAs. Remote SAs may be employed to increase scalability or to optimize performance.

The SA functionality will be developed in three phases:

Fully distributed cellification with local agents only. The filter space will be partitioned into cells and a stream will be associated with each cell. Publishers transmit to streams for cells they are in and receive from cells that lie within their subscription extents. The calculation will be fully distributed, i.e., SAs will not exchange subscription or publication data. Cell density profiles along each dimension will be configurable from the RID. Only local SAs will be supported.

Clustering with local agents only. Subscriptions and publications will be clustered and streams associated with clusters. SAs exchange subscription and publication data and coordinate on stream assignments. Only local SAs will be supported.

*Clustering with local and remote agent*s. Clustering will be performed by remote SAs in cooperation with local SAs.

Transport and discovery services for SAs are provided by the discovery protocol and the consistency protocol.

The SA calculates resource reservations on a stream basis. Resources are reserved through the stream manager.

RTI INITIALIZATION DATA

The RID is a database. It stores and provides an interface to the RTI initialization data. The RTI initialization data consists of the following components:

- FOM-derived data. The RTI obtains all of its knowledge of the FOM contents from the RID.

- RTI optimization and control data. Parameters for configuring the RTI and optimizing its performance are stored in the RID.

The initial implementation reads the RTI initialization data from a local parameter file. Subsequent versions of the RTI may support RID communication and exchange of initialization data between RID instances using stream manager services.

DISTRIBUTION DATABASE

The distribution database (DD) holds run-time data. Notional records are of the form:

{object_id, attribute_name_list, stream, service_type, service_data, ...}

The DD provides accessors to get and set data as needed. Examples include:

- All objects publishing to a stream.
- Attribute names for a given object.
- All streams for a given object.
- Service data for a given object and stream.

The DD is internally organized so as to provide efficient access for the most time critical or intensive access operations.

The DD supports active processing. This includes the ability to register to be called when specific conditions are true or sets of data become available. Examples of such events include:

- Object deleted or created.
- Service changed.
- Attributes in a stream changed.

The DD may be combined with the RID if detailed design indicates substantial benefits.

CONSISTENCY MANAGER

The consistency manager (CM) maintains attributes that are relevant to its client federate in a consistent state. Maintaining a consistent state is different from reliable delivery. Reliable delivery implies some level of recovery of data lost in transmission. Consistency assumes a level of reliable delivery but also must include mechanisms for fetching state that becomes relevant but previously was not. In other words, some form of reliable delivery is necessary for consistent state but is not sufficient. Additional mechanisms are needed to support late joiners, i.e., federates that need to start to receive updates from a stream after an execution has already started.

Support for late joiners is performed in conjunction with consistency mechanisms built into some of the transport services. This task includes:

- Fetching attributes that have become relevant and are not locally cached.
- Deleting attributes that have become irrelevant and are locally cached.

Some of the transport services support consistency mechanisms of their own. For example, the minimum rate service ensures consistency through continual attribute retransmissions. The state consistent service ensures consistency through the heartbeat and request mechanism of the consistency protocol. While both of these services ensure consistency there may be some undesirable latency incurred. Such a latency depends on how a federation chooses to configure protocol parameters. In contrast, the reliable service has no inherent consistency mechanism. Although data lost in transmission will be delivered by the reliable service, there is no provision to accommodate late joiners.

The CM processing to support these requirements is as follows:

Fetching . The CM sends request messages to request the necessary attributes. When a federate needs to receive data from a stream to which it was not previously subscribed, a request message is sent. A request message requests update of all attributes being sent on a particular stream. Upon receiving a request message, the CM requests updates for specific attributes through the client federate's request update service. The CM uses the distribution database to ascertain which objects and attributes are associated with the requested stream. The federate responds by updating the requested attributes.

Deletion. When a federate no longer needs to receive data from a stream the CM invokes its client federate's delete service for objects and attributes associated with the stream. The CM uses the distribution database to obtain a list of objects and attributes being published to the stream. The CM also deletes entries associated with the deleted objects and attributes from the distribution database.

Request message transport. Request messages are sent using stream manager message services. The best effort stream manager service is used if the stream has best effort clients, i.e., is used by the state consistent, minimum rate, or best effort transport services. The reliable stream manager service is used if the stream has reliable clients, i.e., is used by the reliable transport service. Should a particular stream have both best effort and reliable users (e.g., state consistent and reliable transport services) then the particular request message is sent twice: once by best effort and once reliably. The CM ascertains whether a stream is used by best effort or reliable transport services or both through the subscription principal's stream database.

Some federations will not use the reliable service to convey state information – only interactions. Request updates using the reliable service make no sense for these federations. To optimize performance, reliable requests may be disabled through a RID specification. A second optimization that may be employed in the best effort case is to bundle together request messages when there are a large number to be sent at one time. In this case, the composite request is sent to an all hosts stream. This approach reduces the overhead transferring and processing a large number of requests over a short period of time. A third optimization is to incorporate support for a consistency agent to reduce latencies and the scope of protocol interactions [7].

CONSISTENCY PROTOCOL

The consistency protocol (CP) is based on the RITN-developed software [7, 8]. The primary abstraction of CP is the data item. Data items may be created, updated, and deleted. CP supports state consistent reliable multicast data transfer. It guarantees to deliver the most recent state for a data item. Intermediate values may or may not be delivered. This type of reliability is less stringent than a true reliable transport service but also is generally less costly in terms of network and processing overhead. CP uses a heartbeat and NACK (Negative ACKnowledgement) scheme to keep data consistent. CP uses the best effort message service supported by the stream manager.

DISCOVERY PROTOCOL

The discovery protocol (DP) is based on the RITN-developed software. DP provides its clients with the ability to advertise resource that they own, discover resources owned by advertisers, and to monitor the status of advertisers. DP uses the best effort message service supported by the stream manager.

COMMUNICATION SERVICES

The communications services (CS) module represents lower layer communications protocols and interfaces. These include UDP/IP, TCP/IP, and their associated programming and control interfaces.

REFERENCES

[1] Department of Defense High Level Architecture For Simulations, Version 0.5 Interface Specification.

[2] Calvin, James O., Richard Weatherly, “An Introduction to the High Level Architecture (HLA) Run-Time Infrastructure (RTI),” 96-14-103, Fourteenth Workshop on Standards for the Interoperability of Distributed Simulations, March 11-15, 1996.

[3] McGarry, Stephen M., Paul N. DiCaprio, Richard Weatherly, Annette Wilson, “Design Issues for the High Level Architecture (HLA) Run-Time Infrastructure (RTI) Prototype Version 0.2, “ 96-14-104, Fourteenth Workshop on Standards for the Interoperability of Distributed Simulations, March 11-15, 1996.

[4] Calvin, James O., Daniel J. Van Hook, “AGENTS: An Architectural Construct to Support Distributed Simulation,” 94-11-142, Eleventh Workshop on Standards for the Interoperability of Distributed Simulations, September 26-30, 1994.

[5] Calvin, James O., Carol J. Chiang, Daniel J. Van Hook, “Data Subscription,” 95-12-060, Twelfth Workshop on Standards for the Interoperability of Distributed Simulations, March 13-17, 1995.

[6] Van Hook, Daniel J., James O. Calvin, Duncan C. Miller, “RTI Filtering,” Viewgraphs dated January 15, 1996.

[7] Van Hook, Daniel J., James O. Calvin, Joshua E. Smith, “Data Consistency Mechanisms to Support Distributed Simulation,” 95-12-059, Twelfth Workshop on Standards for the Interoperability of Distributed Simulations, March 13-17, 1995.

[8] Van Hook, Daniel J., Eric A. Brittain, “An Implementation of a Data Consistency Mechanism,” 95-13-097, Thirteenth Workshop on Standards for the Interoperability of Distributed Simulations, September 18-22, 1995.